PARALLEL RAM ALGORITHMS FOR COLOURING GRAPHS

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

By

SAJITH G.

to the

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

DECEMBER, 1992

CERTIFICATE

It is certified that the work contained in the thesis entitled **PARALLEL RAM ALGORITHMS FOR COLOURING GRAPHS**, by **SAJITH G**, has been carried out under my supervision and that the work has not been submitted elsewhere for a degree.

S. Saxena

(Dr. Sanjeev Saxena)

Computer Sc. & Engg.

Institute of Technology, Kanpur

1 December 1992

ACKNOWLEDGEMENT

I am very grateful to Dr. Saxena for the invaluable help and guidance that he extended to me, without which this work would have been impossible. I should also thank Dr. Gupta and Dr. Pushpavanam for their helpful suggestions regarding the presentation of the thesis.

And.., my classmates and friends, whose made the past one and a half years of my stay here seem more enjoyable than a vacation.

TABLE OF CONTENTS

ABSTRACT

The following improved parallel algorithms for vertex and edge colouring graphs are suggested in this thesis:

1. An $O(\log^{(k)} n)$ time optimal EREW algorithm for 3-colouring a linked list. The best previous optimal algorithm [CV1] took $O(\log n/\log\log n)$ time on CRCW COMMON PRAM.
2. An $O(\log^{(k)} n)$ time optimal CREW algorithm for $(\alpha+1)$-colouring an α-degree graph for bounded α. This improves the $O(\log n/\log\log n)$ time CRCW COMMON PRAM algorithm of [R].
3. An $O(\log^{(k)} n)$ time optimal CREW algorithm for 3-colouring a bounded-degree graph, which is an improvement over the $O(\log n/\log\log n)$ time CRCW COMMON PRAM algorithm of [R].
4. An $O(\log^{(k)} n)$ time optimal CRCW ARBITRARY algorithm for finding an MIS in an unbounded degree rooted tree. This is the first sublogarithmic time optimal algorithm for this problem.
5. The first optimal logarithmic time algorithm for α-colouring an α-clique-free α-degree graph for bounded $\alpha>2$ on CREW PRAM.
6. An $O(\alpha^2 * \log^2 n)$ time linear processor algorithm for $(\alpha+1)$-edge-colouring an α-degree graph for bounded α. The best previous result, [KS] takes $O(\alpha^7 * \log^2 n)$ time. Besides, the algorithm proposed here is

simpler.

7. An O(log n) time linear processor CRCW ARBITRARY algorithm for 7-colouring a planar graph. This, together with the suggestion of Rajcáni [R] establishes that a planar graph can be optimally 7-coloured in logarithmic time on CRCW ARBITRARY PRAM.

Chapter 1

INTRODUCTION

Algorithms for vertex and edge colouring graphs are studied in this thesis. A vertex colouring of graph G is an assignment of colours to the vertices in such a way that adjacent vertices have distinct colours. We will use natural numbers as our colours. Less informally, a k-vertex colouring of G is a function $\sigma:V(G)\rightarrow\{1,\ldots,k\}$ such that $\sigma^{-1}(j)$ for each j is an independent set (graph theoretic terms are formally defined in section 1.2). Similarly, edge colouring is the problem of finding a function $\pi:E(G)\rightarrow\{1,\ldots,k\}$ for some k, such that, for no two distinct edges e_1 and e_2 incident at the same vertex $\pi(e_1)$ is identical to $\pi(e_2)$.

Chromatic graph theory is mainly the study of these two problems. The origin of this branch of graph theory can be traced back to 1852 when Augustus de Morgan conjectured that every planar graph is 4-colourable. Subsequently, in this century, there has been a flurry of research probing the number of colours required to vertex and edge colour different classes of graphs and the number of ways such colourings can be done.

Graph colouring assumes added significance today due to the various applications it has in diverse fields. So, it has become important to be able to colour graphs

efficiently with as few colours as possible. Some of the applications are given below:

Example 1.1. Register allocation:

In the code generation phase of a typical compilation process, the compiler first generates code assuming that there are as many registers as necessary to hold each of the variables $R_1,\ldots,R_n$ in a separate register. That is, the compiler creates a temporary name (TN) to refer to any temporary storage location. But in practice the number of registers, say k, is fixed. Thus, the problem of binding each TN to some available register is to be solved. This is the classic register allocation problem.

This can be solved using a vertex colouring algorithm as follows: Generate a graph G=(V,E) where V= $\{R_1,\ldots,R_n\}$ and for $R_i,R_j \in V$, $(R_i,R_j) \in E$ iff the scopes in the program of the temporary names R_i and R_j overlap. Find a k-vertex-colouring $\sigma:V\rightarrow\{1,\ldots,k\}$ of G; temporary name v can be allocated to register $\sigma(v)$. ∎

Example 1.2. Scheduling:

Consider the famous class-teacher time-table problem, which is an instance of preemptive open shop scheduling [G]. Here we are required to schedule meetings between teachers and students. The total time student i must meet with teacher j is t_{ij}. A teacher can meet at most one student at a time and vice versa. This can be solved using an edge colouring algorithm. Form a graph (or to be precise, a multigraph) G=(V,E) with the set of

teachers and students taken together as the vertex set. For teacher j and student i there should be t_{ij} edges between vertices i and j in G. A k-edge-colouring of G provides a scheduling which takes a total of k units of time. ■

Example 1.3. Microprogramming:

Microinstructions are often designed to take advantage of the fact that at the microprogramming level, many operations can be performed in parallel. The way of design which thus extensively exploits the parallelism is called horizontal microprogramming. For designing a horizontal microinstruction set, we should divide the set of microoperations into a collection of independent sets. Vertex colouring can be used for this in a manner similar to register allocation problem.■

1.1 Literature on Graph Colouring Algorithms

The chromatic number $\tau(G)$ of a graph G is the minimum k for which G is k-vertex-colourable. Finding the chromatic number of an arbitrary graph is known to be NP hard. This means that the possibility of finding an exact algorithm which can solve reasonably sized problems (say of 60 nodes) in reasonable time (say hours) is very remote. Hence, algorithms which may use more than $\tau(G)$ colours are of interest. If α is the maximum vertex degree of a graph G=(V,E), then G can be easily coloured

sequentially with $(\alpha+1)$ colours in $O(|E|)$[1] time. In parallel setting, Luby [L] and Goldberg and Spencer [GS] have shown that the problem is in NC (i.e., it can be solved in time polylogarithmic in the size of the graph with polynomial number of processors). For the special case of α being bounded, [GPS] gives an $|V|$ processor $O(\log^*|V|)$ time[2] Exclusive Read Exclusive Write (EREW) algorithm whereas, an $O(\log|V|/\log\log|V|)$ time optimal algorithm on the stronger Concurrent Read Concurrent Write (CRCW) model is given in [R].

In an important achievement of graph theory, Brooks showed that any connected graph with a maximum vertex degree of α is α-colourable if it is neither a complete graph on $\alpha+1$ vertices nor a circuit of odd length. An $O(|V|^2)$ sequential algorithm follows from the proof (see e.g., [W]). In parallel setting, the problem is known to be in NC, but the known algorithms are far from being optimal [KN,K,PS]. The most efficient algorithm takes $O(\log^3 n/\log \alpha)$ time with linear processors [PS].

Interestingly, vertex colouring of planar graphs has often received more attention than the general case mentioned above. The four colour conjecture remained an unsolved problem till, in 1977 Appel, Haken and Koch

---

1 $f(n)=O(g(n))$ iff for constants $c>0$ and positive integer n_0, $f(n)< c.g(n)$ for all $n\geq n_0$.

2 All logarithms in this thesis are to base 2.
$\log^{(1)}n=\log n$; $\log^{(k)}n=\log(\log^{(k-1)}n)$ for $k>1$; $\log^*n=\min\{i|\log^{(k)}n\leq 2\}$

settled it by giving an affirmative answer [AHK1,AHK2]. An $O(n^2)$ algorithm can be derived from their proof, but due to its complex nature the algorithm will have astronomical coefficients. So, colouring planar graphs with a fixed and small number of colours becomes significant. More than one O(n) time sequential algorithms for 5-colouring planar graphs are known today [F,Wi]. Coming to parallel algorithms, the best results available are an $O(\log n \log^* n)$ time optimal EREW algorithm for 5-colouring [HCD] and an O(log n) time optimal ARBITRARY CRCW algorithm for 7-colouring [R].

Another class of graphs for which vertex colouring algorithms that use small and constant number of colours and minimum possible resources have been extensively designed are trees and linked lists. These have applications in finding efficient parallel algorithms for many important graph theoretic problems. The parallel symmetry breaking technique of [GPS], provided an $O(\log^* n)$ time algorithm that uses a linear number of processors for 3-colouring a rooted tree. But this algorithm is not optimal. Fastest of the previous optimal 3-colouring algorithms takes O(log n/loglog n) time for lists [CV1] and bounded degree trees [R] and O(log n) time for trees. Attempts to find optimal sublogarithmic time tree colouring algorithms has been on since then (e.g., [R]).

The chromatic index $\tau'(G)$ of a graph G is the minimum

k for which G is k-edge-colourable. If α is the maximum vertex degree of G, obviously $\tau'(G) \geq \alpha$. But Vizing proved that $\tau'(G) \leq (\alpha+1)$. That is the chromatic index of a graph is either α or $\alpha+1$; accordingly, the graph is said to be in class 1 or class 2. Determining the class of an arbitrary graph is known to be NP-hard. But it is possible to $(\alpha+1)$-edge-colour an α-degree graph sequentially in $O(m \sqrt{(n \log n)})$ time (result quoted in [CY]). Whether the problem is in NC is still open; though [KS] proves that the problem is in NC for $\alpha(G)=O(\log^k n)$.

If G is bipartite then, König states in his classical theorem, $\tau'(G)=\alpha$. An $O(m \log n)$ time sequential algorithm is given in [CH]. An efficient parallel algorithm also exists [LPV]. For the case of planar graphs, Vizing proved that, all graphs G with $\alpha(G) \geq 8$ belongs to class 1. He also conjectured that this can be extended to $\alpha(G)= 6,7$ as well. If $2 \leq \alpha(G) \leq 5$ then G can belong to either of the classes. A linear time sequential algorithm as well as an $O(\log^2 n)$ time liner processor EREW algorithm for $\alpha(G) \geq 19$ are described in [CY]. For $9 \leq \alpha(G) \leq 18$, an $O(n \log n)$ time sequential algorithm and an $O(\log^3 n)$ time linear processor EREW algorithm are also given in [CY].

1.2 Basic Graph Theoretic Definitions

* A graph G is an ordered pair of disjoint sets (V,E) such that E is a subset of the set of unordered pair of V. The set V is the set of vertices and E is the

set of edges. If G is a graph then V=V(G) is its vertex set and E=E(G) is its edge set.

* If U is a subset of V (or E), G[U] will denote the subgraph induced by U and G-U will denote G[V-U] (or G[E-U]). For $v \epsilon V$ G-v is G-{v}. For $e \epsilon E$ G-e is G-{e}.

* Two vertices u and v in V are said to be neighbours or adjacent to each other iff $(u,v) \epsilon$ E. Set of neighbours of a vertex v will be denoted by N(v).

* A rooted tree is, a tree in which every non-root node knows its parent.

* And an independent set of G is a subset U of V such that no two vertices in U are adjacent in G. A maximal independent set (MIS for short) of G is an independent set U such that for every vertex v of V, either v is in U or one of its neighbours is in U.

* A vertex colouring $\sigma: V \rightarrow N$ is said to be valid if for every edge $(u,v) \epsilon E$ $\sigma(u) \neq \sigma(v)$; we denote by $\sigma(N(v))$ the set of colours appearing in v's neighbourhood. If $\sigma(v)$ is not defined for every $v \epsilon V$ we say G is partially coloured. A colour c is said to be feasible at v if c is not in $\sigma(N(v))$.

* For the problem of α-vertex-colouring, an uncoloured vertex in a partially coloured graph is said to be at impasse if it has no feasible colour in $\{1,..,\alpha\}$. If v is a vertex at impasse, its neighbour of colour i will be denoted by v_i $(i=1,2\cdots,\alpha)$.

* An α-β component in G is a component of the subgraph

induced by vertices coloured α or β. Note that interchanging colours α and β in an α-β component does not affect the validity of the colouring. So, for a vertex v at impasse if v_α and v_β belong to different α-β components interchanging colours in one of them will resolve the impasse.

* A graph in which every vertex has degree at most α, will be called an α-degree graph.
* A graph in which every vertex has degree exactly α, is an α-regular graph. A 3-regular graph is also called a cubic graph. A subcubic graph has a maximum vertex degree of 3.
* An α-clique is a complete graph on α vertices. Note that it is $(\alpha-1)$-regular.

1.3 Overview of Thesis

Chapter 2 describes the various parallel RAM models used in this thesis.

An $O(\log^{(k)} n)$ time optimal EREW list-colouring algorithm is presented in chapter 3. This improves the result of [CV1]. Consequently, improved algorithms are obtained for the problems of colouring bounded degree graphs with a constant number of colours, 3-colouring a bounded degree rooted tree and finding an MIS in a rooted tree. The previous best algorithms for the two former problems were due to Rajcáni [R] took and O(log n/loglog n) time, whereas the latter had only an O(log n) time algorithm.

Chapter 4 gives the first logarithmic time optimal algorithm for Brook's colouring a bounded degree graphs.

An $O(\log^2 n)$ time linear processor edge-colouring algorithm for bounded degree graphs is discussed in chapter 5.

Some nontrivial details not mentioned by Rajcáni in his logarithmic time optimal algorithm for 7-colouring planar graphs are presented in chapter 6.

Chapter 2

PARALLEL RAM MODELS

A (sequential) random access machine (RAM) models a one accumulator computer in which instructions are not permitted to modify themselves. A RAM consists of a read only input tape, a write only output tape, a program and a memory. Input tape is a sequence of squares each of which holds an integer (possibly negative). Both the input tape and the output tape can be scanned only in one direction. The memory consists of registers $r_0, r_1, \ldots$ each of which is capable of holding an integer of arbitrary size. The program for RAM is not stored in memory, and uses an instruction set that resembles those usually found in real computers. (A typical instruction set consists of LOAD, STORE, ADD, SUB, MULT, DIV, READ, WRITE, JUMP, JGTZ, JZERO and HALT instructions). To specify the time and space complexity of RAM programs we use the unit cost criterion where each RAM instruction requires one unit of time and each register requires one unit of space.

All the parallel computation models used here belong to the family of parallel random access machines (PRAM). A PRAM has **p** synchronous RAMs all having access to a common memory.

* An exclusive-read exclusive-write (EREW) PRAM does not allow simultaneous access by more than one

processor to the same memory location for read or write purposes.

* A concurrent-read exclusive-write (CREW) PRAM allows simultaneous access for reads but not writes.
* A concurrent-read concurrent-write (CRCW) COMMON PRAM allows concurrent access for both reads and writes, but a write attempt is considered valid only if all the processors are trying to write the same value.
* In ARBITRARY CRCW PRAM model processors are allowed to simultaneously attempt to write different values in the same memory location and an arbitrary one of them is guaranteed to succeed; algorithms should work correctly,independent of the processor which succeeds.

If Seq(n) is the worst-case running time of the fastest known sequential algorithm for a problem, an optimal parallel algorithm for the same problem runs in O(Seq(n)/p) time using p processors.

Chapter 3

$O(\log^{(k)} n)$ TIME OPTIMAL LIST COLOURING ALGORITHM AND ITS APPLICATIONS[1]

In this chapter an $O(\log^{(k)} n)$ time optimal EREW PRAM algorithm for three-colouring a linked list is first obtained in section 3.1. This algorithm is then used to design optimal $O(\log^{(k)} n)$ time CREW PRAM algorithms for colouring bounded-degree graphs with a constant number of colours in section 3.2 and bounded-degree rooted trees with 3 colours, in section 3.3. Recall that, the best previous optimal algorithms for both these problems took $O(\log n/\log\log n)$ time [R]. Yet another application of the list colouring algorithm is an $O(\log^{(k)} n)$ time optimal CRCW ARBITRARY algorithm for finding an MIS in a general rooted tree, this is studied in section 3.4. The basic algorithm uses a generalized version of list contraction technique of Anderson and Miller [AM].

3.1 Three-Colouring Linked Lists

The following generalisation of the list contraction technique of [AM] will be used in the algorithm.

Lemma 3.1 (Lemma 2. of [B]): For any n, r, $k \in N$ with $1 \le r \le n$ and $\log^{(k)} n \le r$, a linked list of length n can be reduced in $O(kr)$ time using $O(kn)$ operations to a linked list of length at most n/r on an EREW PRAM. ∎

---

1 The results of this chapter are reported in [SS1].

Procedure 3-Colour-Lists

Input: A linked list L of length n.

Output: A 3-colouring of L.

Algorithm:

Step 1: Using $r=\log^{(k)}n$ in the above lemma reduce the length of L to $n/\log^{(k)}n$ in $O(\log^{(k)}n)$ time using $O(n)$ operations for some constant k. Each vertex in the reduced list is guaranteed to have a processor attached to it [AM].

Step 2: Apply the 3-colouring algorithm of [GPS] to the reduced list. This takes $O(\log^{*}n)$ time with $O(n*\log^{*}n/\log^{(k)}n)=O(n)$ operations.

Step 3: Reconstruct L from the reduced list by following the operations of step 1 in reverse order. Since no two adjacent vertices were simultaneously removed at any time in step 1 each vertex inserted back has both its neighbours coloured and hence can be given the least colour not in its neighbourhood. Time and processor bounds are the same as for step 1.

Thus the following theorem,

Theorem 3.1: A linked list can be 3-coloured in $O(\log^{(k)}n)$ time using $n/\log^{(k)}n$ processors on an EREW PRAM. ∎

3.2 Colouring bounded degree graphs

Theorem 3.2: A graph in which the degree of any vertex is at most a constant α can be $(\alpha+1)$-coloured in $O(\log^{(k)} n)$ time using $n/\log^{(k)} n$ processors on a CREW PRAM.

Proof: Assume we are given a graph G each of whose vertices is of degree at most α. As in [DK], the edge set E of graph G is fully decomposed in constant time into $t=O(\alpha)$ sets, $E_1,..,E_t$, where each set defines a list graph [see proof of lemma 2.2, page 207 of DK].

Each of these lists can be three coloured in $O(\log^{(k)} n)$ optimal time, by theorem 3.1.

Lists thus three coloured can be combined to obtain an $\alpha+1$ colouring of G as follows:

Iteratively for i:= 1 to t, return E_i to the graph, each time recolouring the vertices to maintain a consistent colouring (as in [GPS]). When E_i is added the existing $(\alpha+1)$-colouring may become inconsistent. Now, each vertex has two colours - one the colour it had at the end of the previous iteration and one from the colouring of $G[E_i]$. The pairs of colours form a valid $3(\alpha+1)$-colouring of the graph. The iteration finishes by enumerating the colour classes, recolouring each vertex of the current colour with the minimum colour not in its neighbourhood. We can recolour all the vertices of the same colour in parallel because they are independent. ∎

3.3 Three-Colouring a Bounded Degree Rooted Tree

Theorem 3.3: A rooted tree T in which the degree of each vertex is at most a constant α, can be 3-coloured in $O(\log^{(k)} n)$ time using $n/\log^{(k)} n$ processors on a CREW PRAM.

Proof: Find an $(\alpha+1)$-colouring $\sigma:V(T)\rightarrow\{0,\ldots,\alpha\}$ of T as in the proof of theorem 3.2. From this a 3-colouring can be obtained in constant time using $O(n)$ operations as described below [GPS,R].

```
for i := α downto 3 do
begin
    for each v∈V in parallel do
        if v is not a root then σ(v) := σ(parent(v))
        else σ(v):=c where c∈ {0,1,2} and c≠σ(v)
    for each v with C(v)=i in parallel do
        σ(v):=c where c∈ {0,1,2} and c∉ σ(N(v))
end.
```

The theorem clearly follows. ∎

3.4 Finding MIS in a Rooted Tree

Theorem 4.1: An MIS of a general rooted tree can be found in $O(\log^{(k)} n)$ time using $O(n)$ operations on an ARBITRARY CRCW PRAM.

Proof: Let T=(V,E) be the rooted tree. For $v\in V$, let p[v] be the parent of v in T. For every $v\in V$ arbitrarily select one child s[v]. Form a new rooted tree T'=(V,E') where E'

contains only those edges of E which are between a vertex and its selected child. Now T' is a collection of linked lists. Find a 3-colouring σ of T' in $O(\log^{(k)} n)$ optimal time.

```
for v∈V do in parallel
    if v≠s[p[v]] then p'[v]:="undefined"
    else p'[v]:=p[v]
M'=φ ;
for v∈V with p'[v] defined do in parallel
begin
    if p'[v] is a root in T' add v to M'.
       for i in {0,1,2} do if σ(v)=i and neither v nor
    its neighbours in T' are  already in M' then add v
    to M'
end;
```

As no root in T' can be in M', an MIS M of T can be obtained by adding to M' each isolated vertex v of T' with $p[v] \notin M'$.

Resource requirements of the algorithm are dominated by time to find 3-colouring of linked lists. Thus the theorem follows. ∎

Chapter 4

BROOKS' COLOURING BOUNDED DEGREE GRAPHS[1]

This chapter describes an O(log |V|) time optimal CREW algorithm for α-colouring an $(\alpha+1)$-clique-free α-degree graph, $\alpha>2$ being bounded. An algorithm for 3-colouring subcubic graphs is described in section 4.1. This algorithm is used as "basis" for deriving the required general algorithm in section 4.2.

4.1. 3-Colouring Subcubic Graphs

In this section, we describe an optimal algorithm for vertex colouring 3-degree graphs. For simplicity of discussion we assume that the graph is cubic. Generalising to include subcubic graphs is straight forward.

Procedure 3-Colour-Cubic-Graphs

Input: A cubic graph G=(V,E) in adjacency list representation; |V|=n.

Output: A 3-colouring σ:V→{1,2,3} of G.

High Level description of the algorithm:

First a maximal independent set (MIS) M of G is found and removed. Vertices of the remaining graph H are coloured 1 or 2, except for one and only one vertex in each odd cycle of H, which is left uncoloured. Vertices of

M are coloured 3. Each uncoloured vertex has all the three colours in its neighbourhood and hence is at impasse. Observe that for each vertex v at impasse, v_1 and v_2 are the end points of a simple 1-2 chain $P_{12}(v)$ in G.

For each v at impasse do the following:

1) If v_1 and v_3 are in different 1-3 components of G then resolve the impasse at v by interchanging colours 1 and 3 in one of the two 1-3 components; if this is not possible then, after this step, there will be a simple 1-3 chain in G with v_1 and v_3 as end points.

For each vertex v at impasse now there is a 1-3 chain $P_{13}(v)$ with v_1 and v_3 as its end points.

2) If v_2 and v_3 are in different 2-3 components of G then resolve the impasse at v by interchanging colours 2 and 3 in one of the two 2-3 components; if this is not possible then, after this step, there will be a simple 2-3 chain in G with v_2 and v_3 as end points.

3) Due to interchanges made in steps (1) and (2) above, $P_{12}(v)$ and $P_{13}(v)$ now need not be respectively 1-2 and 1-3 components of G. If either case holds then we resolve the impasse at v through local recolouring of $P_{12}(v)$ or $P_{13}(v)$ as is required. So we are left with only the situation where v_i and v_j are the end points of a simple i-j path $P_{ij}(v)$ of G for each vertex v at impasse and $1 \le i < j \le 3$.

4) If v_2 is adjacent to both v_1 and v_3 then

Recolour v, v_1, v_2 and v_3 with colours 1, 2, 3 and 2 respectively.

else

Interchange colours 1 and 3 in $P_{13}(v)$. The resulting graph is similar to the case discussed in (3) and is solved using a similar technique.

Algorithm in detail:

Step 1:

a) Obtain an MIS M of G and for each v in M let $\sigma(v)=3$. Remove M from G to get H, i.e., H=G-M. /* Vertices of the MIS are coloured 3 */

Remark MIS is found by first finding a 4-colouring C:V→{1,2,3,4} of G using the algorithm for $(\alpha+1)$-colouring an α-degree graph. Then, for each i:=1 to 4, in turn add $w \epsilon V$ to M, if C(w)=i and no neighbour of w is already in M. The $(\alpha+1)$-colouring can be found in O(log n) time with n/log n processors on a CREW PRAM, by dividing the graph into $O(\alpha)$ sets of disjoint chains (as in [DK]), 2-colouring each set using the list ranking algorithm (of say, [AM]) and finally combining these colourings (as in [GPS]).

As every vertex in G is adjacent to some vertex in M, degree of each vertex in H is at least one less than its degree in G; thus maximum degree of H is two, and hence H consists of disjoint chains and cycles.

b) For each vertex of H, find whether it is in a chain or a cycle. Assuming we have p processors, this is done by first reducing the size of H to p using the list

contraction technique of [AM] and then applying the recursive doubling step O(log p) times. During the recursive doubling step, for each $v \epsilon V(H)$, we can also find the smallest numbered vertex F(v) in the contracted version of the component containing v.

Remark Observe that for $u,v \epsilon V(H)$ $F(u)=F(v)$ if and only if u and v belong to the same connected component of H.

c) Let $J= \{ v \mid v=F(u)$ and u is in a cycle in H $\}$ and $H'=H-J$. That is, from every cycle of H, exactly one vertex is missing in H', which hence is a collection of disjoint chains. For each chain of H' select one of its two end points as the "tail" (say the one having higher index). Use the list ranking algorithm of [AM] to compute the distance of each vertex from the tail of the chain to which it belongs. Give vertices of odd rank colour 1 and even rank colour 2.

Further, for each even cycle C of H, the vertex $v_e(C)$ of C which is in J is coloured 2. Each of the remaining vertices in J, belongs to an odd cycle of H, is at impasse, and is left uncoloured. Let I be the set of impasse vertices.

Remark Observe that the colouring is valid as, of the three neighbours in G of $v_e(C)$, the two in H are of colour 1 and the one in M is of colour 3.

d) For $v \epsilon V$ do in parallel

if v is coloured 1 or 2 and v belongs to an odd cycle of H then P(v):=F(v)

else P(v):="undefined"

Remark Every odd cycle of H provides a path between v_1 and v_2, if v is the impasse vertex contained in it. Moreover, each vertex on this path is coloured 1 or 2 and its neighbour outside the path is coloured 3. Thus, v_1 and v_2 belong to the same 1-2 component, which is a simple path with v_1 and v_2 as its end points. For each vertex w on this path, P(w)=v. In the subsequent steps, the vertices for which P is defined will be referred to as P-vertices. Others are non-P-vertices. Note that every P-component, i.e., a component in the subgraph induced by P-vertices, is a chain. We will be extensively using the following procedures:

```
Procedure RESOLVE
begin
    For v∈I do in parallel
        if v has at least one colour missing in its
        neighbourhood then give v the minimum feasible
        colour and remove it from I.
end.
Procedure UPDATE(S)
/* S will be either P or Q; Q will be defined later. In
general S can be any partial function S:V→V */
begin
    For v∈V do in parallel
        if S(v) is defined but S(v) is not at impasse
        then S(v):="undefined".
```

end.

Step 2:

a) /* if the neighbourhood of v contains colours 1 or 3 only, then recolour v with colour 2 */

For $v \epsilon V$ do in parallel

begin

if $\sigma(v)=1$ and $\sigma(N(v))=\{3\}$ then $\sigma(v):=2$;

if $\sigma(v)=3$ and $\sigma(N(v))=\{1\}$ then $\sigma(v):=2$;

end

Call RESOLVE;

Call UPDATE(P);

Remark If a vertex v is at impasse, then v_3 can have at most one neighbour coloured 1 and v_1 can have at most one neighbour coloured 3. Moreover, as every vertex w coloured 1 or 3 has a neighbour coloured 2, every 1-3 component of G is a simple path. Thus, each of v_3 and v_1 is an end point of a 1-3 chain. That is every impasse vertex has exactly two (not necessarily distinct) 1-3 chains going out of it.

b) Find a maximal set of 1-3 components such that no two of them touch the same impasse vertex (for details see appendix at the end of the chapter). Interchange colours 1 and 3 in these components.

Remark Impasse is resolved for a vertex if colour was changed in any one of the two 1-3 components emanating out of it.

c) Call RESOLVE;

Call UPDATE(P);

Remark For each $v \epsilon I$, now we have a simple 1-3 path in G with v_1 and v_3 as its end points (see appendix at the end of the chapter). But, note that, now it is not necessary for v_1 and v_2 to be in the same 1-2 component, let alone a 1-2 path.

d) For $v \epsilon V$ do in parallel

Q(v):="undefined" /* initialise */

For $v \epsilon I$ do in parallel

For each w in the 1-3 path from v_1 to v_3 do in parallel Q(w):=v.

Remark In the subsequent steps, the vertices for which Q is defined will be referred to as Q-vertices.

Step 3: /* Repeat step 2 for colours 2 and 3 */

a) If the neighbourhood of v contains colours 2 or 3 only, then recolour v with colour 1, in a manner similar to step 2(a). As result, the 2-3 subgraph of G is a 2-degree graph and each vertex at impasse has exactly two (not necessarily distinct) 2-3 chains touching it.

b) Find a maximal set of 2-3 components such that no two of them touch the same impasse vertex (for details see appendix) Interchange colours 2 and 3 in these components.

c) Call RESOLVE;

Call UPDATE(P); Call UPDATE(Q);

Remark For each $v \epsilon I$, now we have a simple 2-3 path in G

with v_2 and v_3 as its end points. But, v_1 and v_2 may not be in the same 1-2 component of G. And similarly, v_1 and v_3 may not be in the same 1-3 component of G.

Step 4:

a) Using the list ranking algorithm, identify all P-components of G (each of which is a chain, by step 1d).

b) for each chain L_p identified in 4(a) in parallel do

begin

/* Let v be the impasse vertex associated with L_p; v_1 and v_2 are the end points of L_p */

i) Adjust the ranks in L_p such that v_2 has rank 1. For each $x \epsilon L_p$, predecessor of x, $pred_p(x)$ (respectively successor of x $succ_p(x)$) is the lower (higher) ranked P-neighbour of x. $Pred_p(v_2)$ and $succ_p(v_1)$ are left "undefined".

ii) Find the lowest ranked vertex w in L_p such that w has a non-P neighbour coloured the same as $succ_p(w)$; if there is no such vertex let w be "undefined".

Remark Note, that if w is defined then $w \neq v_1$. Moreover as v_2 is an end point of a 2-3 chain, $succ_p(v_2)$ is coloured 1. So, $w \neq v_2$, and rank(w)>1.

iii) if w is defined then

begin

for each $x \epsilon L_p$ with rank(x)< rank(w) do

$\sigma(x) := \sigma(succ_p(x))$

Give w the only colour feasible there.

end.

Remark Each vertex from v_2 to $pred_P(w)$, gets the colour of its successor. Now, w has exactly two colours in its neighbourhood, one the colour of $succ_P(w)$ (which is same as the colour of its non-P neighbour) and the other its own previous colour. Thus, w can be given a new colour and the colouring remains valid. Moreover, vertex v_2 retains its colour, but v_1 gets recoloured 2. Thus, v has two neighbours coloured 2, and impasse is resolved.

iv) /* w is "undefined" */ Find the lowest ranked vertex u in L_P such that u's non-P neighbour is not coloured 3; if there is no such vertex let u be undefined.

Remark $1<rank(u)\leq rank(v_1)$ (See the remark after step ii).

v) If u is defined then

begin

For each $x\epsilon L_P$ with $rank(u)\leq rank(x)<rank(v_1)$ do

$\sigma(x):=\sigma(succ_P(x))$

Give a feasible colour to v_1.

end.

Remark If u is defined then $pred_P(u)$ has a non-P neighbour coloured 3; so $\sigma(pred_P(u))\neq 3$. As w is undefined, $\sigma(u)\neq 3$ (otherwise $pred_P(u)$ would have been w). That is, both u and $pred_P(u)$ are coloured from {1,2}. Let the non-P-neighbour of u be coloured $c\epsilon\{1,2\}$. Then, $\sigma(u)\neq c$ and $\sigma(pred_P(u))=c$. Also, by a similar argument, if $succ_P(u)$ is defined, it is coloured 3. Thus, the colouring remains

valid and v_2 gets a new colour. As, v_1 and v_3 are still coloured 1 and 3 respectively, impasse at v is resolved.

In each of the unresolved cases, both u and w are undefined. That is, every vertex in L_P has a non-P neighbour of colour 3, or in other words, L_P is a 1-2 chain.

c) Call RESOLVE;

Call UPDATE(P); UPDATE(Q);

Step 5: /* Repeat step 4 for colours 1 and 3 */

Using the list ranking algorithm, identify all Q-components of G and for each chain L_Q identified thus (with v as its associated impasse vertex) in parallel do begin

i) Adjust the ranks in L_Q such that v_1 has rank 1. For each $x \epsilon L_Q$, $pred_Q(x)$ (respectively $succ_Q(x)$) is defined iff $x \neq v_1$ ($x \neq v_3$) and is x's lower (higher) ranked Q-neighbour.

ii) Find the lowest ranked vertex w in L_Q such that w has a non-Q neighbour coloured the same as $succ_Q(w)$; if there is no such vertex let w be "undefined".

iii) if w is defined then

begin

for each $x \epsilon L_Q$ with rank(x)< rank(w) do

$\sigma(x) := \sigma(succ_Q(x))$

Give w the only colour feasible there

end.

iv) Find the lowest ranked vertex u in L_Q such that u's non-Q neighbour is not coloured 2; if there is no such vertex let u be undefined.

v) If u is defined then

begin

For each $x \epsilon L_Q$ with $rank(u) \leq rank(x) < rank(v_3)$ do

$\sigma(x) := \sigma(succ_Q(x))$

Give a feasible colour to v_3.

end.

end.

Call RESOLVE; Call UPDATE(P); UPDATE(Q);

Remark So, we are left with only the case where v_i and v_j are the end points of a simple i-j path $P_{ij}(v)$ of G for $1 \leq i < j \leq 3$.

Step 6:

/* For $v \epsilon I$ let $N(v)=\{x,y,z\}$. Also let $\sigma(x)=1$ $\sigma(y)=2$ and $\sigma(z)=3$; i.e., $v_1=x$ $v_2=y$ and $v_3=z$. $P_{12}(v)$ is a path of P-vertices with x and y as its end points. Let n_x be the neighbour of x in $P_{12}(v)$; n_x is coloured 2 (n_x may be $v_2=y$) */

For $v \epsilon I$ do the following:

a) If $v_2(=y)$ is adjacent to both $v_1(=x)$ and $v_3(=z)$ then

begin /* $(x,z) \notin E$ because G has no 4-cliques */

$\sigma(x) := \sigma(z) := 2;$

$\sigma(y) := 1;$ $\sigma(v) := 3;$

end.

b) /* else */ Interchange colours 1 and 3 in $P_{13}(v)$.

Remark Now, it is not necessary for every vertex in $P_{12}(v)-\{x\}$ to have its non-P neighbour coloured 3. So, m_x, the non-P neighbour of n_x can be coloured either 1 or 3. But, if x and y are adjacent (i.e., $n_x=y$) then (see step a) y and z can not be adjacent (i.e., $m_x \neq z$), and m_x is coloured 3.

c) If $\sigma(m_x)=1$ then /* m_x is coloured 1 */

begin /* $n_x \neq y$ */

swap colours between x and n_x

/* i.e., $\sigma(x):=2$; $\sigma(n_x):=3$ */

$\sigma(v):=3$

end.

Remark Before this step, n_x had two neighbours of colour 1. Also, x had no neighbour other than n_x coloured 2. That is, $G[\{x,n_x\}]$ is a 2-3 component of G. Hence, the colouring remains valid.

d) Else /* $\sigma(m_x)=3$; n_x may be y */

i) Rank $P_{12}(v)$ beginning at $y=v_2$.

ii) Let w be the minimum ranked vertex in $P_{12}(v)$ whose non-P neighbour and successor in $P_{12}(v)$ are of the same colour.

/* w exists because at least n_x satisfies this condition. So, $rank(w) \geq rank(n_x)$. If n_x is y then w is also y */

iii) for each $t \epsilon P_{12}(v)$ with $rank(t) < rank(w)$ do

$\sigma(t):=\sigma(t's\ successor)$

iv) Give w a new feasible colour. /* That is, y gets colour 1. Impasse at v is resolved because, x and z are still coloured 3 and 1 respectively. */

Remark Now a call to procedure RESOLVE will solve the problem.

Theorem 4.1: A 4-clique free subcubic graph can be 3-coloured in O(log n) time with linear processor-time product.

Proof: We prove that the procedure 3-Colour-Cubic-Graphs 3-colours a 4-clique free cubic graph in O(log n) time with linear processor-time product.

Correctness of the algorithm is obvious from the remarks following individual steps.

With n/log n processors step 1(a) can be done in O(log n) time (see the remark after step 1 a). Rest of the procedure is dominated by a constant number of invocations to the list ranking algorithm which can be solved in O(log n) optimal time [AM]. Hence the claim on resource requirements.

For any subcubic graph on n vertices, a cubic graph on O(n) vertices of which the former is a subgraph, can be created in constant time. Hence the theorem.∎

Corollary If a subcubic graph G contains 4-cliques $\Sigma_1, \Sigma_2, \cdots, \Sigma_k$, then $G-\Sigma_1-\Sigma_2-\cdots-\Sigma_k$ can be coloured using three colours and each of $\{\Sigma_1, \Sigma_2, \cdots, \Sigma_k\}$ with four colours

in O(log |V|) optimal time on a CREW PRAM.

Proof In a degree 3 graph, vertices in a 4-clique are not adjacent to any other vertex. Hence the 4-cliques can be identified in O(α)=O(1) time using O(|V|) processors. ∎

4.2 α-Colouring α-Degree Graphs

In this section the general problem of colouring an α-degree graph with α colours is considered. For simplicity of discussion we assume that the graph is α-regular, extensions to general case are straight forward.

Procedure α-Colour-α-Regular-Graphs

Input: An (α+1)-clique-free α-regular graph G=(V,E) in adjacency list representation. |V|=n.

Output: $\sigma: V \rightarrow \{1,..,\alpha\}$

Step 0:

If $\alpha \leq 3$ use procedure 3-Colour-Cubic-Graphs of the previous section.

Step 1:

a) Obtain an MIS M of G and for each v in M let $\sigma(v)=\alpha$. Remove M from G to get H, i.e., H=G-M. /* Vertices of the MIS are coloured α */

Remark The maximum vertex degree of any vertex in H is (α-1). Observe that H need not be (α-1) colourable as it may contain α-cliques; but a vertex in a clique cannot be adjacent to a vertex not in the clique, i.e., all cliques are isolated (connected) components. Moreover these

α-cliques can easily be identified in $O(\alpha)=O(1)$ time.

b) From every α-clique add the smallest numbered vertex to I.

c) Delete vertices of I from H to get H', i.e., H'=H-I

Remark As H' does not contain any α-clique, it is α-1 colourable.

d) Recursively (α-1)-colour H'; I is the set of impasse vertices in G.

Remark For each $v \epsilon I$, $G[\{v, v_1, \ldots, v_{\alpha-1}\}]$ is an α-clique.

e) For each $v \epsilon I$, select the minimum $\beta \epsilon \{1,2,\cdots,\alpha-1\}$ such that v_α and v_β are not adjacent. Swap colours between v_1 and v_β.

Remark This ensures that v_1 and v_α are not adjacent. Observe that, v_α is not adjacent to all of $\{v_1, .. v_{\alpha-1}\}$ as v_α is adjacent to v and G does not contain an (α+1)-clique.

Step 2:

a) For $v \epsilon V$ do in parallel

begin

if v is coloured 1 and ((at least three of its neighbours are coloured α) or (at least two of its neighbours are coloured α and at least one of its neighbours is at impasse)) then

recolour v with the colour missing in its neighbourhood

if v is coloured α and ((at least three of its

neighbours are coloured 1) or (at least two of its neighbours are coloured 1 and at least one of its neighbours is at impasse)) then

recolour v with the colour missing in its neighbourhood

end.

Call RESOLVE; /* see the previous section */

Remark After this step, every vertex coloured α or 1 is adjacent to at most two vertices coloured 1 or α (respectively), thus every 1-α component of G is a chain or a cycle. Also, for each $v \epsilon I$, v_1 and v_α are end points of 1-α chains.

b) Find a maximal set of 1-α components such that no two of them touch the same impasse vertex (for details see appendix). Interchange colours 1 and α in these components. Call RESOLVE.

Remark Impasse is resolved for a vertex if colours in either of the two 1-α components emanating out of it were swapped in this step. For each $v \epsilon I$, now we have a simple 1-α path in G with v_1 and v_α as its end points.

c) For $v \epsilon V$ do in parallel

P(v):="undefined" /* initialise */

For $v \epsilon I$ do in parallel

For each w in the 1-α path from v_1 to v_α do in parallel P(w):=v.

Remark As before, a vertex with P defined is called a P-vertex; the graph induced by P-vertices is a set of

disjoint chains. Also, we will make use of the procedure UPDATE described in the previous section.

Step 3:

Repeat 2(a) and 2(b) for colours 2 and α.

Call UPDATE(P);

Remark Now v_2 and v_α are the end points of the same 2-α path, for every $v \epsilon I$. But v_1 and v_α need not even be in the same 1-α component.

Step 4:

Using the list ranking algorithm, identify all P-components (chains) of G. For every P-component L (with v as its associated impasse vertex) do the following /* resolve impasses as in step 3 of section 4.1 */.

i) Adjust the ranks in L so that v_α has rank 1.

ii) Find the lowest ranked vertex w such that, w has a non-P neighbour coloured the same as its successor. If there is no such w let w be "undefined".

iii) If w is defined then

begin

for each $x \epsilon L$ with rank(x)< rank(w) do

$\sigma(x) := \sigma$(x's successor).

give a feasible colour to w.

end.

iv) /* w is "undefined" */ Find the lowest ranked vertex u in L such that u has a non-P neighbour

coloured 1 or α; if there is no such vertex let u be "undefined". /* $u \neq v_\alpha$ */

v) If u is defined then

begin

For each $x \epsilon L$ with $rank(u) \leq rank(x) < rank(v_1)$ do

$\sigma(x) := \sigma(x\text{'s successor})$.

give a feasible colour to v_1.

end.

Call RESOLVE; Call UPDATE(P);

Remark So, we are left with only the case where u is undefined. As result, v_1 and v_α are the end points of the same 1-α path $P_1(v)$ and v_2 and v_α are the end points of the same 2-α path $P_2(v)$. It is quite possible that $P_1(v)$ and $P_2(v)$ may have vertices in common. By step 2, v_1 and v_α are not adjacent, hence $P_1(v)$ does not degenerate into a single edge. But, $P_2(v)$ may be a single edge.

Step 5:

For $v \epsilon I$ do in parallel

If the α-coloured neighbour s of v_1 is on $P_2(v)$ then /* s is on both $P_1(v)$ and $P_2(v)$ and as v_1 and v_α are not adjacent, $s \neq v_\alpha$ */

begin

Recolour s with another feasible colour /* as s is on both $P_1(v)$ and $P_2(v)$, it has two neighbours of colours 1 and 2 each */

Recolour v_1 with α /* Impasse at v is resolved */

$\sigma(v):=1$.

end

else /* s is not in $P_2(v)$ */

i) Interchange colours 2 and α in $P_2(v)$.

ii) Resolve the impasse at v by recolouring v_1 with 2 ($\sigma(v_1):=2$).

Remark After interchange in step (i) v_2 is of colour α and no neighbour of v_1 is of colour 2 (recall, v_α and v_1 are not adjacent). Thus, v_1 can be given colour 2, resolving the impasse at v.

Theorem 4.2: An α-degree Brooks' graph can be α-coloured in O(log n) time with n/log n processors; where $\alpha>2$ is a constant.

Proof: From the proof of theorem 1 it follows that steps 2 through 9 of the above procedure takes only O(log n) time with n/log n processors. That is an instance of α-colouring can be reduced to one of $(\alpha-1)$-colouring in as much resource bounds. Hence the procedure α-Colour-α-Regular-Graphs α-colours an α-regular Brooks' graph in O(log n) time with n/log n processors; where $\alpha>2$ is a constant. The theorem follows as in in the proof for theorem 1.

Corollary If an α-degree graph G contains $(\alpha+1)$-cliques

$\Sigma_1, \Sigma_2, \cdots, \Sigma_k$, then $G-\Sigma_1-\Sigma_2-\cdots-\Sigma_k$ can be coloured using α colours and each of $\{\Sigma_1, \Sigma_2, \cdots, \Sigma_k\}$ with $(\alpha+1)$ colours in $O(\log |V|)$ optimal time on a CREW PRAM.

Proof Similar to corollary to theorem 1.

APPENDIX: Finding a maximal independant set of α-β components

Assume that, every α-β component in G is a simple path and for every vertex v at impasse v_α and v_β are end points of α-β chains. It is required to find a maximal set of α-β components in graph G, such that no two members in the set "touch" the same impasse vertex; a component Γ touches a vertex v if there is a vertex $w \epsilon \Gamma$, such that v and w are neighbours.

Form a graph G1 in which each vertex corresponds to an α-β component of G. An edge is placed between two vertices of G1 if and only if the α-β components corresponding to them touch the same impasse vertex. Since an α-β component can touch at most two impasse vertices, one at each end, the maximum vertex degree of G1 is 2. Remove all isolated vertices and self loops from G1.

Find the connected components of G1 (as in step 1 of algorithm in section 3) and identify each as a chain or a cycle. From each cycle remove a vertex (again as in step 1(d) of section 4.1). Let the resulting graph be G2. Using list ranking algorithm find ranks of all vertices in G2. For every odd ranked vertex of G2 interchange colours α

and β in the corresponding α-β component of G.

An α-β component in G is a component of the subgraph induced by vertices coloured α or β. Note that interchanging colours α and β in an α-β component does not affect the validity of the colouring, whether the colouring be partial or complete. So, for a vertex v at impasse if v_α and v_β belong to different α-β components interchanging colours in one of them will resolve the impasse.

Chapter 5

EDGE COLOURING BOUNDED DEGREE GRAPHS

An $O(\alpha^2 * \log^2 n)$ time linear processor EREW algorithm for $(\alpha+1)$-edge-colouring an α-degree graph for bounded α, is given in this chapter. The algorithm of [KS] shows that the problem is in NC when $\alpha = O(\log^k n)$. When α is a constant their algorithm also takes only $O(\alpha^7 * \log^2 n)$ time. But, the algorithm proposed here has a much simpler proof; moreover the exponent of α is smaller here.

If maximum degree of any node in the given graph G is α, then by Vizing's theorem, G can be edge coloured with $\tau'(G) \leq (\alpha+1)$ colours.

The proof of Vizing's theorem [MG] uses induction on the number of edges in the graph. Every α-degree graph with α edges is obviously $(\alpha+1)$-colourable. This forms the basis. Now assume that every α-degree graph with m-1 edges is $(\alpha+1)$-edge-colourable. Let $G=(V,E)$ be an α-degree graph with m edges. Then, G-e, for any $e \epsilon E$ has an $(\alpha+1)$-edge-colouring. A colour c is said to be free at $v \epsilon V$ if none of the edges incident at v is coloured c. As we use $(\alpha+1)$ colours to colour G-e, at least one colour is free at each vertex of G. Let $v_0, v_1 \epsilon V$ be the end vertices of e. Assume that no colour j is free at both v_0 and v_1, otherwise, e can be given colour j.

Now, consider the following data structure:

Definition: A (c_0, c_k)-fan, $F=(v_0, v_1, \ldots, v_k)$ for distinct

$v_1,\ldots,v_k \in N(v_0)$ satisfies following conditions:

(i) colour c_i is missing at v_i for $0\leq i\leq k$,

(ii) the edge (v_0,v_i) is coloured c_{i-1} for $1<i\leq k$ and

(iii) either c_k is missing at v_0 or $c_k=c_j$ for $0<j<k$.

The first vertex v_0 is referred to as head of the fan F and is denoted by head(F) and the last vertex v_k is called tail of F and is denoted by tail(F).

The fan is said to be maximal, if for no vertex v_{k+1}, $F'=(v_0,v_1,\ldots,v_k,v_{k+1})$ is a (c_0,c_{k+1})-fan.

Assuming adjacency list representation for the graph and one processor for each node, the following procedure can find the fan for every node at which an uncoloured edge is incident:

```
Procedure create-fan(v0)
If (v0,v1) is uncoloured for some v1∈N(v) then
begin
     Add v0,v1 to fan(v) /* fan(v) is not complete */
     i:=2.
     While fan(v) is not complete do
     begin
          find vi such that (v0,vi) is coloured ci-1
          if vi∉fan(v) then add vi to fan(v)
          if ci is free at some vj∈fan(v) j≠i then
               fan(v) is complete and k:=i
          else i:=i+1
     end.
end.
```

With O(n) processors the above procedure takes only $O(\alpha)$ time if the edge list of each vertex is given in an array indexed with colours and if each edge list entry knows the colours free at its both ends. These conditions can again be met if each vertex informs each member of its edge list the colours free at it, the edge list of each vertex is compacted using recursive doubling and then finally the compacted array is sorted. Clearly, this again can be done in $O(\alpha)$ time.

If c_k is missing at v_0 (in this case the fan is called local), edge (v_0,v_i) can be given colour c_i for $1\leq i\leq k$, thus completing the proof. (This is called the fan operation). So, assume that $c_k=c_j$ for $0<j<k$. Then at least one of the two c_0-c_k paths originating at c_k and c_j, does not contain v_0. Interchange colours c_0 and c_k in this path. (This is the chain operation). Now we have a fan $(v_0,\ldots,v_1)$ with c_1 free at v_0; i.e., a fan operation completes the proof.

Observation 1 : The fact that there is a colour $d_1\neq c_1$ free at v_1 is is of no relevence in the above proof.

Procedure edge-colour-bounded-degree-graphs

input: An α-degree graph $G=(V,E)$ in adjaceny list representation; $|V|=n$, $|E|=m$. Assume without loss of generality that G is α-regular.

Output: An $(\alpha+1)$-edge-colouring $\pi:E\rightarrow\{1,\ldots,\alpha+1\}$ of G.

Step 1: If G is not Eulerian add a vertex u that is

adjacent to every odd degree vertex in G. Let G' be the augmented graph thus obtained. If G is Eulerian let G'=G.

TIME O(log n) with n EREW processors.

Remark Since the number of odd degree vertices in any graph is even, G' is Eulerian.

Step 2: Find an Eulerian tour in G'. If G≠G' let this tour start at s=u, otherwise, let it start at some sϵV. Label the edges of the tour $e_1, e_2 . e_3, \ldots$ in order. Form two graphs $G_1=(V,E_1)$ and $G_2=(V,E_2)$ where E_1 and E_2 are $\{e_i | i$ is odd$\} \cap E$ and $\{e_i | i$ is even$\} \cap E$ respectively.

TIME As Eulerian tour in any Eulerian graph G can be found in O(log n) time using O(n+m) processors on CRCW PRAM [AV], it can also be found in $O(\log^2 n)$ time on EREW PRAM. As for bounded degree graph $m \leq \alpha n/2$, number of processors used is O(n).

Remark If G≠G' or |E| is even then both G_1 and G_2 are ceil(α/2)-degree graphs. If G=G' and |E| is odd then vertex s has degree (α/2)+1 in G_1 and (α/2)-1 in G_2. Every vertex other than s has degree (α/2) in both G_1 and G_2.

Step 3: This step is done only if s has degree (α/2)+1 in G1. Select and remove from G_1 an edge e=(s,t) for some tϵN(s). Now both G_1 and G_2 are ceil(α/2)-degree graphs. Recursively (ceil(α/2)+1)-colour them. Exactly one colour is free at s. Insert e back in G1. Call create-fan(t). By Obsrvation.1 we can perform chain and fan operations as required and give e a colour in {0,..,α}.

TIME Chain and fan operations can be done in O(log n)

time.

Remark Both G_1 and G_2 are (ceil(α/2)+1)-coloured.

Step 4: Add ceil(α/2)+1 to every colour used in G_2.

Remark Now, at worst, we have an (α+3)-colouring of G. Let E_c be the set of edges coloured c.

Step 5: Form a conflict graph $C=(E_{\alpha+2},\Phi)$ where $(e,f)\epsilon\Phi$ for some $e,f\epsilon E_{\alpha+2}$ iff there is a path of length at most 2 between e and f in G. The maximum vertex degree δ of C is $O(\alpha^2)$. Find a (δ+1) colouring σ of C. Let E_i^σ denote the set of edges e in $E_{\alpha+2}$ such that $\sigma(e)=i$.

TIME $O(\alpha^4 * \log^* n)$ with n processors.

Step 6: For i:=1 to (δ+1) do serially the following:

a) For each $e=(u,v)\epsilon E_i^\sigma$ in parallel do

If u<v then call create-fan(u) else call create-fan(v).

b) For each local fan created above, perform the fan operation.

Remark Now we can have (c,d)-fans in G for $1\leq c,d\leq\alpha$, $c\neq d$. The class of a fan F is (c,d), c<d if F is a (c,d) or (d,c)-fan. For particular values of c and d, a constant fraction of the set of (c,d) and (d,c)-fans Σ_{cd} can be solved using the following procedure:

```
Procedure SOLVE(c,d)
begin
    1. For each FεΣcd let rep(F) be the first vertex in
       F at which d is free and the c-d path starting at
       it doesn't touch head(F). We say a c-d path ends at
```

F if either rep(F) or head(F) is an end point of it.

2. Form a conflict graph H in which each vertex corresponds to a c-d path of G and $(x,y) \epsilon E(H)$ iff the c-d paths corresponding to x and y end at the same fan in G. Obviously, H is a 2-degree graph. From every odd cycle of H remove one vertex. Rank the remaining graph, and for the vertices of odd rank intechange colours c and d in the corresponding c-d paths of G. Perform fan operation on every (c,d)-fan that has now turned local. At least $2/3*|\Sigma_{cd}|$ number of fans are now solved.

3. For every $F \epsilon \Sigma_{cd}$ still unsolved, there is a c-d path connecting head(F) and rep(F). If rep(F)=tail(F) then create a new fan for F.

end.

Note that SOLVE(c,d) may change classes of fans which it doesn't solve. That is, we have an instance of the following scheduling problem:

There are given n tasks, which can be divided into a constant number, say k of classes. A constant fraction of each class can be completed in O(log n) time with n processors. It is required to schedule the n tasks on n processors.

An obvious solution is to find the class with the maximum cardinality and solve a constant fraction of it. So, the entire set of tasks can be completed in $O(\log^2 n)$

time with n processors.

c) While E_i^{σ} has edges coloured $\alpha+2$ do

begin

Find the largest cardinality class (c,d) of fans.

SOLVE(c,d);

end.

TIME The loop of step 6 is executed $O(\alpha^2)$ times and by the discussion in 6(b) each iteration takes $O(\log^2 n)$ time.

Remark All edges in $E_{\alpha+2}$ are now given one of the $(\alpha+1)$ colours.

Step 7: Repeat steps 5 and 6 for $E_{\alpha+3}$

Theorem 5.1: An α-degree graph, for bounded α, can be $(\alpha+1)$-coloured in $O(\alpha^2 * \log^2 n)$ time with n processors on an EREW PRAM.

Proof: Time for $(\alpha+1)$-edge-colouring an α-degree n-vertex graph by the above algorithm is,

$$T(\alpha,n) = T(\alpha/2,n) + O(\alpha^2 * \log^2 n)$$
$$= T(1,n) + O(\alpha^2 * \log^2 n)$$
$$= O(\alpha^2 * \log^2 n)$$

Chapter 6

OPTIMAL LOGARITHMIC TIME 7-COLOURING FOR PLANAR GRAPHS

Finding an optimal logarithmic time algorithm for colouring a planar graph with a small and fixed number of colours had been open for a long time, until recently Rajcáni [R] as a passing remark suggested that an optimal O(log n) time CRCW ARBITRARY algorithm, that uses seven colours, exists.

We can remove and 7-colour the subgraph H of G induced by vertices of degree 6 or less using Rajcáni's O(log n/loglog n) time optimal algorithm for colouring an α-degree graph with $(\alpha+1)$-colours (for bounded α). (Instead, theorem 3.2 can also be used). Once 7-colourings are obtained for both H and G-H, vertices of H are added back to G-H in the order of their colours. Since every vertex v inserted back has at most 6 neighbours and all of them are coloured, v has at least one feasible colour (the one not present among its neighbours). This obviously can be done in constant time with n processors. Since, the number of vertices in G of degree 6 or less is at least $n/6$[1], this means that, an n-sized instance of the problem

---

1 For a connected planar graph G=(V,E) with $|V|=n$ and $|E|=m$, by Euler's formula, $m \leq 3n$. So, if n_i is the number of vertices of degree i, for $i \geq 1$, we have, $\Sigma_{1\leq i}(i * n_i) \leq 6 * \Sigma_{1\leq i}(n_i)$

i.e., $\Sigma_{7\leq i}((i-6) * n_i) \leq 5n_1 + 4n_2 + 3n_3 + 2n_4 + n_5$

i.e., $\Sigma_{7\leq i}(n_i) \leq 5 * (n_1+\cdots+n_6)$; hence, $\Sigma_{1\leq i}(n_i) = n \leq 6 * (n_1+\cdots+n_6)$

can be reduced to an atmost-5n/6-sized instance in O(log n/loglog n) time with O(n loglog n/log n) processors. So, Rajcáni claims that the following framework of Hagerup [H] can be used to find an O(log n) time optimal algorithm:

Lemma 6.1 (follows from Lemma 1 of [H]):

Let **P** be a class of problem instances, each of which has an integer, size, associated with it. Suppose that,

(1) problem instances in **P** of size n can be solved in O(log n) time with n processors.

(2) There is a real constant $c < 1$ and an integer constant n_0 such that any $p \in$ **P** of size $n \geq n_0$ can be reduced in O(log n/loglog n) time with O(n loglog n/log n) processors to a problem instance $\Phi(p)$ of size atmost n.

Then problem instances in **P** of size n can be solved in O(log n) time with n/log n processors. ∎

But, clearly, for Rajcáni's claim to be true, we need an O(log n) time linear processor algorithm for 7-colouring planar graphs. Unfortunately, such an algorithm is not mentioned in the available literature; neither does [R] give any reference.

Such an algorithm is described in the next section thus completing the claim in [R].

6.1 O(log n)-time Linear Processor Algorithm for 7-Colouring Planar Graphs

Assume that the input graph G is in adjacency list representation and one processor each is associated with vertices and edges of the graph. The algorithm, in the i^{th} iteration, removes from the current graph G_i, the subgraph induced by vertices of degree 6 or less along with the processors assigned to them, until G_i is empty. Since, for every planar graph at least a constant fraction of nodes are of degree 6 or less, the number of iterations will be bounded by O(log n). Removing a subgraph can be done in O(1) time on CRCW ARBITRARY PRAM. 7-colouring of the removed subgraph is done immediately after its removal, and by theorem 3.2 this takes only $O(\log^{(k)} n)$ optimal time. That is, the total time taken is $O(\log n)+O(\log^{(k)} n)= O(\log n)$ only!

The graph is reconstructed by treating the subgraphs in the reverse order of their removal. In a subgraph itself, vertices are added to G in the order of their colours. Each vertex inserted back is coloured with the minimum colour feasible at it. Again the time taken is O(log n). So, we have the following theorem:

Theorem 6.1: A planar graph G=(V,E) can be 7-coloured in $O(\log |V|)$ time using $O(|V|/\log |V|)$ CRCW ARBITRARY processors.■

CONCLUSIONS

Summary

In this thesis following results were proved:

1. A linked list can be optimally 3-coloured in $O(\log^{(k)} n)$ time on an EREW PRAM.
2. A bounded degree graph can be coloured with a constant number of colours in $O(\log^{(k)} n)$ optimal time on a CREW PRAM.
3. A bounded degree rooted tree can be 3- coloured in $O(\log^{(k)} n)$ optimal time on a CREW PRAM.
4. An MIS of a general rooted tree is obtained in $O(\log^{(k)} n)$ optimal time on a CRCW ARBITRARY PRAM.
5. A bounded degree graph can be Brook's coloured in $O(\log n)$ optimal time on a CREW PRAM.
6. For bounded α, an α-degree graph can be $(\alpha+1)$-edge coloured in $O(\log^2 n)$ time with $O(n)$ EREW processors.
7. A planar graph can be 7-coloured in $O(\log n)$ optimal time on a CRCW ARBITRARY PRAM.

Suggestion for Further Work

It appears that technique used in [CV1] for 3-colouring a list in $O(\log n/\log\log n)$ time, can be generalised to get an $O(\log n/\log\log n)$ time algorithm for

tree-colouring as well.

The best known optimal algorithm for 3-colouring unbounded-degree-trees takes O(log n) optimal time [GPS] (unbounded maximum degree). It first forms a 7-colouring of the tree and then reduces it to a 3-colouring. Recall that the O(log n/loglog n) time algorithm of [R], as well as the one given in theorem 3.3, 3-colours only a bounded-degree rooted tree. Following procedure "appears" to solve this problem:

As in [GPS] form a (loglog n)-colouring of the tree T, which in particular is a (log n/loglog n) colouring. The vertices are bucket-sorted on colour. Now, recolour the vertices in the order of their colours by giving each one the minimum colour not in its neighbourhood. Assuming that there are (n * loglog n/log n) processors it is easy to show that the time taken is O(log n/loglog n) or that the number of operations is O(n).

But, unfortunately, here it is not necessary that every vertex will have exactly two colours in its neighbourhood.

One way out may be to simultaneously recolour every vertex with the colour of its parent. This, ensures that the first set of vertices which is recoloured will have exactly two colours in its neighbourhood, but it is not clear how this can also be maintained for remaining steps.

REFERENCES

[AM] R.J.Anderson and G.L.Miller, Deterministic Parallel List Ranking, Algorithmica, **6:6** (1991) 859-868.

[AH1] K.I.Appel, W.Haken and J.Koch, Every Planar Map Is Four Colourable. PartI: Discharging, Illinois J. Math., **21** (1977) 429-490.

[AH2] K.I.Appel, W.Haken and J.Koch, Every Planar Map Is Four Colourable. PartII: Reducibility, Illinois J. Math., **21** (1977), 491-567.

[AV] M.Atallah and U.Vishkin, Finding Euler Tours in Parallel, J. Of Computer and Systems Sciences, **29**, 330-337 (1984).

[B] P.C.P.Bhatt, K.Diks, T.Hagerup, V.C.Prasad, T.Radzik and S.Saxena, Improved Deterministic Parallel Integer Sorting, Inform. and Comput. **94** (1991) 29-47.

[CY] M.Chorbak and M.Yung, Fast Algorithms for Edge Colouring Planar Graphs, J. Algorithms, **10**, 35-51,

(1989).

[CV1] R.Cole and U.Vishkin, Faster Optimal Parallel Prefix Sums and List Ranking, Inform. and Comput. **81** (1989) 334-352.

[CV2] R.Cole, U.Vishkin, Deterministic Coin Tossing with Applications to Optimal Parallel List Ranking, Information and Control, **70:1** (1986) 32-53.

[DK] N.Dadoun and D.G.Kirkpatrick, Parallel Processing for Efficient Subdivision Search, ACM Symp. on Computational Geometry (1987) 205-214.

[F] G.N.Frederickson, On Linear Time Algorithms For Five-Colouring Planar Graphs, Inform. Process. Lett., **19**, (1984), 219-224.

[G] T.Gonzalez, A Note on Open Shop Preemptive Schedules, IEEE Trans. Comput., **C-28** (1979) 782-786.

[GPS] A.V.Goldberg, S.A.Plotkin and G.E.Shannon, Parallel Symmetry Breaking in Sparse Graphs, Proceedings, 19th Annual ACM Symposium on Theory of Computing (1987) 315-324.

[GS] M.Goldberg and T.Spencer, A new parallel algorithm for the maximal independant set problem, SIAM J. Comput. **18:2** (1989) 419-427.

[H] T.Hagerup, Optimal Parallel Algorithms on Planar Graphs, Information & Computation, **84**, 71-96, 1990.

[HCD] T.Hagerup, M.Chorbak and K.Diks, Optimal Parallel 5-colouring of Planar Graphs, SIAM J. Comput., **18:2**, 288-300, April 1989.

[KN] M.Karchmer and J.Naor, A fast parallel algorithm to colour a graph with Δ colours, J. of Algorithms, **9** (1988) 83-91

[K] H.J.Karloff, An NC algorithm for Brook's Theorem, Theoretical Computer Science **68:1** (1989) 89-103.

[KS] H.Karloff and D.Shmoys, Efficient Parallel Algorithms for Edge Colouring Problems, J. Algorithms **7** (1985) 39-52.

[L] M.Luby, A fast parallel algorithm for maximal independant set problem, SIAM J. Comput. **15:4** (1986) 1036-1053.

[MG] J.Misra and D.Gries, A Constructive Proof for

Vizing's Theorem, Inform. Process. Lett., **41** (1992) 131-133.

[MS] C.A.Morgenstern and H.D.Shapiro, Heuristics for Rapidly Four-Colouring Large Planar Graphs, Algorithmica, **6:6**, 1991, 869-891.

[PS] A.Panconesi and A.Srinivasan, Improved Distributed Algorithms for Colouring and Network Decomposition Problems, in: 24th Annual ACM Symp. Th. Comput (1992).

[R] P.Rajcáni, Optimal parallel 3-colouring for rooted trees and applications, Inform. Proc. Letters **41** (1992) 153-156.

[SS1] Sajith.G and S.Saxena, Optimal Parallel Algorithms for Colouring Linked Lists and Trees, TR-CS-149, IIT Kanpur (1992). Manuscript submitted for publication.

[SS2] Sajith.G and S.Saxena, Optimal Parallel Algorithm for Brooks' Colouring Bounded Degree Graphs in Logarithmic Time witout Concurrent Writes, TR-CS-158, IIT Kanpur (1992) Manuscript submitted for publication.

[W] R.J.Wilson, Introduction to Graph Theory (Longman, London, 1979).